Accelerating integer-based fully homomorphic encryption using Comba multiplication

Accelerating Integer-based Fully Homomorphic Encryption using Comba Multiplication

Ciara Moore, Máire O'Neill, Neil Hanley, Elizabeth O'Sullivan
Centre for Secure Information Technologies
Queen's University Belfast
{cmoore50, maire.oneill, n.hanley, e.osullivan}@qub.ac.uk

***Abstract*—Fully Homomorphic Encryption (FHE) is a recently developed cryptographic technique which allows computations on encrypted data. There are many interesting applications for this encryption method, especially within cloud computing. However, the computational complexity is such that it is not yet practical for real-time applications. This work proposes optimised hardware architectures of the encryption step of an integer-based FHE scheme with the aim of improving its practicality. A low-area design and a high-speed parallel design are proposed and implemented on a Xilinx Virtex-7 FPGA, targeting the available DSP slices, which offer high-speed multiplication and accumulation. Both use the Comba multiplication scheduling method to manage the large multiplications required with uneven sized multiplicands and to minimise the number of read and write operations to RAM. Results show that speed up factors of 3.6 and 10.4 can be achieved for the encryption step with medium-sized security parameters for the low-area and parallel designs respectively, compared to the benchmark software implementation on an Intel Core2 Duo E8400 platform running at 3 GHz.**

I. Introduction

Fully Homomorphic Encryption (FHE) is a recently developed type of encryption scheme, introduced by Gentry [1], which enables computations on data while it remains in an encrypted form. A somewhat homomorphic encryption (SHE) scheme is firstly created, which allows a *limited* number of additions and multiplications of ciphertexts. This is extended using the techniques, such as squashing and bootstrapping, proposed by Gentry in [1] to create a FHE scheme, which supports *unlimited* multiplications and additions of ciphertexts. Random noise is generated with each operation, in particular with multiplications, and there are various methods proposed to manage this noise as it grows with each computation, such as modulus switching, which has been proposed by Brakerski *et al.* [2]. An application of FHE, for example, is secure computation on the cloud whereby users could take advantage of the cloud computing platform, without having to disclose data to a third party service provider. Another application of SHE and FHE is within multi-party computation, and schemes for this purpose have been proposed [3], [4].

FHE schemes have greatly advanced over recent years, since its introduction in 2009 [1]. There has been a vast array of work in the theoretical domain [2], [5]–[13], however current schemes are not yet efficient enough for real-time applications. For example, an evaluation of AES using FHE is reported to take around 36 hours on a machine with 256 GB RAM [10]; in another software implementation of the FHE scheme proposed by Gentry and Halevi, bitwise encryption at the highest security level is stated to take 3 minutes and the public key sizes required in this scheme range from 17 MB to 2.25 GB [5].

Thus, to address this shortcoming, optimised architectures that target alternative platforms, such as Graphics Processing Units (GPUs) and FPGAs, have recently been proposed [14]–[19]. Wang *et al.* [14] implemented Gentry and Halevi's FHE scheme [5] on the NVIDIA C2050 GPU and achieved speed up factors of around 7 compared to the original implementation at a small security level. Furthermore, a design for a large-number multiplier for FHE targeting a Stratix-V FPGA technology was presented last year by Wang *et al.* [15]; this multiplier uses the FFT algorithm and is reported to be twice as fast as the same multiplication implemented on the NVIDA C2050 GPU.

Previously, Cao *et al.* proposed the first hardware implementation [16] of the encryption step of an FHE scheme over the integers [11], specifically investigating the use of a Virtex-7 FPGA platform. The experimental results reported a speed improvement factor of 11.25 for an implementation on a Virtex-7 XC7VX980T FPGA of the encryption step using the large security parameters compared to the original software implementation [11]. An extended version of this work is available on the IACR ePrint Archive [20] and includes optimised hardware architectures for the encryption step of the two integer-based FHE schemes [11], [12].

Architectures targeting Application Specific Integrated Circuit (ASIC) technology have also been proposed to improve the performance of FHE schemes [17], [18]. Doröz *et al* presented a custom hardware architecture for a million-bit multiplier for the implementation of Gentry and Halevi's FHE scheme [5], and estimates show similar performance to the original software implementation. All of the previous hardware and GPU implementations to date employ the Fast Fourier Transform (FFT) to perform the large scale multiplications required for these FHE schemes.

The objective of this work is to design an optimised architecture for the implementation of the encryption step of the FHE scheme over the Integers specifically tailored to a FPGA device. Unlike previous work which used the FFT for the large multiplication operations, an alternative method for large integer multiplication is used in our work, which

utilises the high speed DSP multiplication blocks (DSP48E1s) available on Xilinx Virtex-7 FPGAs to implement the Comba multiplication scheduling method [21]. This builds on previous work on the use of DSP slices and Comba multiplication for the hardware acceleration of FHE [19], where a hardware architecture for a Comba multiplier was proposed and estimated timings were given for the large integer multiplier required in the encryption step of an integer-based encryption scheme [12]. Rather than estimating timings, this work presents implementation and synthesis results for a Xilinx Virtex-7 FPGA of two optimised hardware architectures for the encryption step.

A Xilinx Virtex-7 FPGA is targeted in particular in this work because of the high suitability of FPGAs for DSP applications. Virtex-7 FPGAs contain many DSP slices, each offering dedicated (25 x 18)-bit multiplication and 48-bit accumulation, which can run at frequencies of up to 741 MHz [22]. Moreover FPGAs are reconfigurable which allows for fast prototyping and testing.

There are several types of FHE schemes, which have developed from the original lattice-based schemes proposed by Gentry [1], [5], [6]. More recent schemes are based on the learning with errors and ring learning with errors problems, such as [2], [8]–[10]. Another type of FHE scheme is the proposed FHE over the integers, introduced by van Dijk *et al.* [7] and extended by Coron *et al.* [11], [12]. FHE over the integers has been selected as the target FHE scheme in our work, because of its comparable performance to other FHE schemes, and additionally there have been further advancements in the theoretical domain, which have improved the performance by minimising the size of the public keys required [12] and with the use of batching techniques [13]. Moreover, the designs proposed include building blocks, such as large integer multiplication, which are integral to most FHE schemes, such as Gentry and Halevi's FHE scheme [5].

The remainder of this paper is organised as follows: Section II gives an overview of FHE over the integers with particular focus on the encryption step to be implemented; the Comba multiplication technique is outlined in Section III; Section IV details the Barrett modular reduction method used in the implementations; in Sections V and VI the two novel architectures for the FHE encryption step are presented; synthesis results are detailed in Section VI, along with a comparison to previous implementations and finally, Section VII concludes the paper.

II. Fully Homomorphic Encryption over the Integers

FHE over the Integers was introduced in 2010 by van Dijk *et al.* [7]. This type of FHE scheme is relatively simpler than other FHE schemes and is based on the Approximate GCD problem: given several x_i, where $x_i = p \cdot q_i + r_i$, find the secret key p. The original FHE scheme over the integers [7] was subsequently extended by Coron *et al* [11], [12], where in particular the public key sizes were reduced, through the use of pseudo random number generation.

TABLE I: Parameter sizes for encryption step in (1)

Param. Sizes	**Bit-length,** x_i	**Bit-length,** b_i	τ
Toy	150k	936	158
Small	830k	1476	572
Medium	4.2m	2016	2110
Large	19.0m	2556	7695

This work focuses on the encryption step in the scheme proposed in [12]. The encryption step is one of several steps within the scheme and contains two important building blocks, large integer multiplication and modular reduction. Although we initially focus solely on the encryption step in this work, the construction and optimisation of these building blocks required in the encryption step will also be used in future work to implement the other steps within this FHE scheme, and moreover can also be used to implement other FHE schemes.

The encryption step is defined as

$$c \leftarrow m + 2r + 2\sum_{i=1}^{\tau} x_i \cdot b_i \; mod \; x_0 \tag{1}$$

where $m \in \{0, 1\}$ is the message bit; r is a random noise parameter; x_i are integers generated from the public key as described in the key generation step in [12]; b_i are randomly selected integers, which are much smaller than the x_i. The parameter sizes are given in Table I. The selection of suitable parameters is out of the scope of this current work; for more information on the security levels, parameter selection and for detail on the rest of the FHE scheme, see the original work by van Dijk *et al* [7] and Coron *et al* [11], [12].

III. Comba Multiplication

As can be seen from Equation (1), a large multiply-accumulate is required, which is the main bottleneck in the encryption step. As mentioned in Section II, multiplication is also needed in other steps and in other FHE schemes in the literature, hence the work is transferable. Currently, the approach taken by the research community is to use the Fast Fourier Transform for fast multiplication, which is suitable for very large multiplication sizes.

An alternative fast multiplication method is Karatsuba multiplication [23] , which is asymptotically faster than traditional schoolbook multiplication. It requires intermediate values to be stored for each multiplication, and hence is not very suitable for an FPGA implementation of these FHE schemes, as they require very large multiplications and the storage of intermediate values would be problematic.

Therefore, we take an alternative approach and make use of the fast embedded multiplication blocks available within the DSP slices on Xilinx Virtex-7 FPGAs and combine this with the Comba scheduling method [21]. As can be seen from Table I, the x_i are much greater than the b_i. To minimise the number of read and write operations, the multiplication block width w is set to the next power of two greater than or equal to the the b_i bit length. Thus, the multiplication block width

is 1024, 2048, 2048 and 4096 for the four parameter groups respectively.

Comba multiplication is a scheduling method to effectively control the multiplication and accumulation of partial products [21]. It reduces the number of expensive write accesses to memory compared to traditional school book multiplication. For example, if the Comba scheduling method is used to multiply two large integers x and y, these integers are divided into several smaller words; thus x and y have n_0 and n_1 words respectively, and these are multiplied and accumulated to calculate the large integer multiplication of x and y. Instead of writing the $n_0 \times n_1$ partial products to memory, as necessary for schoolbook multiplication, only $n_0 + n_1 - 1$ partial products are required when the Comba scheduling method is used. Following Algorithm 1, after each partial product multiplication-accumulation step the least significant word is written to memory and the remainder is shifted and then accumulated with the next partial product.

Algorithm 1: Comba partial product accumulation

Input: n_0-word x, n_1-word y, where $n_0 \leq n_1$
Output: $(n_0 + n_1 - 1)$ partial products, pp_i

1: **for** i in 0 to $(n_0 + n_1 - 2)$ **do**
2: **if** $n_0 < i$ **then**
3: $pp_i = \sum_{k=0}^{i-1}(x_k \times b_{i-k})$
4: **else**
5: $pp_i = \sum_{k=0}^{n-1}(x_k \times b_{i-k})$
6: **end if**
7: **end for**

return pp_i

Güneysu demonstrates a parallelised Comba multiplication method, which takes advantage of the available DSP blocks on an FPGA [24]. The accumulation within the Comba multiplication is also carried out using the dedicated accumulator available in each DSP slice. We take a similar approach for the multiplications required in Equation (1).

IV. Barrett Modular Reduction

The most commonly used modular reduction methods are the Montgomery and Barrett methods. Modular reduction involves division and hence is a costly operation in hardware. We use an improved Barrett reduction method in our design rather than Montgomery, which requires expensive pre-processing and post-processing to and from the Montgomery domain. This lends itself more to repeated reduction operations, such as in an exponentiation operation, which Equation (1) does not require.

Barrett reduction is used to carry out the modular reduction required in the encryption step in Equation (1). The reduction algorithm uses two multiplications and a subtraction. The same multiplication block will be used for both the multiplication and the modular reduction in the proposed low-area design, as these operations are sequential. This minimises the hardware area usage.

The Barrett Reduction method is optimised, as proposed by Dhem [25], so that only one subtraction is needed after the multiplication, when $\alpha \geq m$ and $\beta \leq -2$, as described in Algorithm 2. This algorithm was also previously used in the FPGA implementation of the encryption step of integer-based FHE [16]; however as mentioned, this previous work used the FFT algorithm rather than the Comba multiplication algorithm as used here.

Algorithm 2: Barrett reduction [25], [26]

Input: x, m-bit p, α, β and a precomputed constant $p_1 = \lfloor 2^{m+\alpha}/p \rfloor$
Output: $y = x \, mod \, p$

1: $\sigma = \lfloor \frac{\lfloor x/2^{m+\beta} \rfloor \times p_1}{2^{\alpha-\beta}} \rfloor$;
2: $\hat{p} = \sigma \times p$;
3: $y_1 = x - \hat{p}$ and $y_2 = y_1 - \hat{p}$;
4: **if** $y_2 < 0$ **then**
5: $y = y_1$
6: **else**
7: $y = y_2$
8: **end if**

return y

V. Hardware Architecture for Encryption Step

The main building block in the encryption step (1) to be implemented is the multiplication-accumulation unit. Two architectures are proposed; both architectures contain a finite state machine, controlling the data unit and the read and write operations. The designs both use off-chip memory to store input operands, intermediate values and final results. The main difference between the two architectures is the number of multiplication blocks: the first design contains just one muliplication block, minimising resource usage, and the second design contains several multiply-accumulate blocks, maximising speed.

The DSP slices available on Xilinx Virtex-7 FPGAs are used within the multiplication blocks in both designs. Figure 1 is a structural overview of a DSP slice, as given in the Xilinx 7 Series DSP48E1 Slice User Guide [27]. Both a 25×18- bit signed multiplier and a 48-bit accumulation unit are offered in each slice. Thus, a 16-bit unsigned multiplier is used in each slice in the multiplication blocks in both designs in this section. Each block consists of $\frac{w}{16}$ DSP slices, where w is the width of the overall multiplication in each block and is given in Table II.

The target of this work is an optimised architecture and implementation of the encryption step introduced in Section II, in order to improve its performance. Thus, it is assumed that there is enough off-chip memory available to store input operands, intermediate values and final results. This assumption is reasonable, as the designs proposed in this work use 64-bit read and write operations and the FPGA can access shared memory, for example with a PC, using a high speed PCI bus.

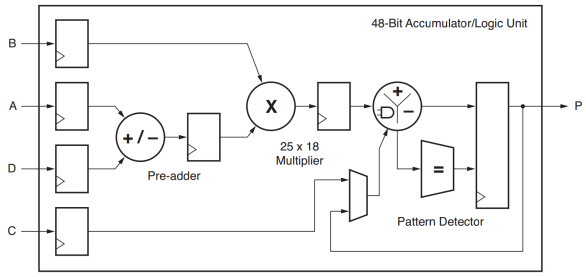

Fig. 1: Xilinx 7 Series DSP48E1 Slice [27]

A. Low-area Architecture Design

Figure 2 depicts the architecture of the first proposed design, which aims to minimise the area resource required. The main element in this architecture is one multiplication block of width w, approximately the size of the smaller multiplicand b_i, which is controlled by a finite state machine. The same multiplication block is also repeatedly used within the modular reduction step. The design therefore minimises area usage, but obviously this somewhat limits the performance. The multiplexers control the adders for two uses: addition required for the multiplication-accumulation operation (when $acc\_sel = 1$) and secondly subtraction required in the modular reduction operation (when $acc\_sel = 0$).

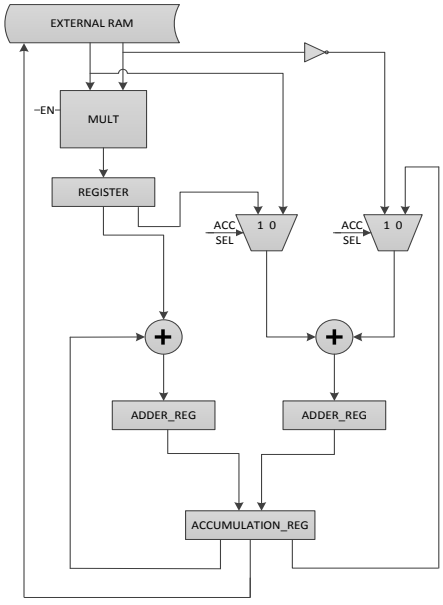

Fig. 2: Low-area architecture design of encryption step

B. Parallel Architecture Design

The goal of the second design is speed. The maximum possible number of multiply-accumulate blocks that can fit on a high-end Virtex-7 FPGA are used in this architecture in order to maximise the performance of the encryption step. The Virtex-7 FPGA XC7VX980T is targeted because it has the largest number of DSP slices and a large amount of logic cells. The parallel architecture is outlined in Figure 4 and uses multiple multiply-accumulate blocks. The architecture of an individual multiply-accumulate block is illustrated in Figure 3. The number of blocks depends on the size of the multiplier w, listed in Table II, which is the size of next power of two greater than or equal to the number of bits in the smaller operand in the multiplication block b_i. As the size of the b_i operand increases with increasing parameter security levels, from the smallest toy security setting to the large security parameter setting, the number of multiplication blocks that can fit on the FPGA decreases.

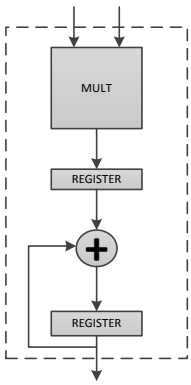

Fig. 3: Multiply-accumulate block design

VI. Results

The two proposed architectures were implemented using the Xilinx ISE Design Suite 14.1 synthesis tool. The target device is the Virtex-7 XC7VX980T-2FFG1926, the Virtex-7 FPGA with the largest number of DSP slices. The synthesis results and hardware area usage are given in Table II and Table III for the parallel and low-area designs respectively. These designs fit comfortably on the target FPGA device. Moreover, the low-area design achieves high clock frequencies of around 300 MHz for the four parameter security sizes, which are much greater than the clock frequencies achieved in the implementation of the high-speed architecture. It should be noted that post place and route of the designs will give slightly worse results; however the speed of the read and write operations can be greatly improved upon, with the use of external memory interfaces and for example DDR3 SDRAM memory, which can run at four times the clock frequency [28].

As expected, the hardware resource utilisation is much greater for the implementation of the encryption step using

TABLE II: Synthesis results for the high-speed parallel architecture

Param. Sizes	w, Multiplication block bit-width	Parallel Mul-Acc Blocks	Freq. (MHz)	DSP48E1s	Slice Reg	Slice LUTs
Toy	1024	20	197.465	1344	469577	262054
Small	2048	10	192.806	1408	491712	311914
Medium	2048	10	197.758	1408	497773	313104
Large	4096	5	197.758	1536	542949	365315

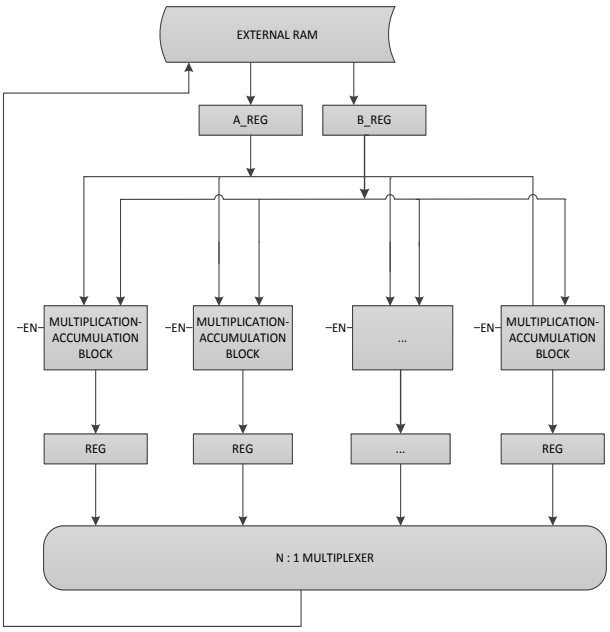

Fig. 4: Parallel architecture design of encryption step

the high-speed architecture. It is to be noted that it is possible to fit more multiply-accumulate blocks on the target-device than the number stated in Table II, because the target FPGA has 3600 available DSP48E1 slices and 612000 Slice LUTs. However the synthesis frequencies decrease with the addition of multiply-accumulate blocks and eventually slow the performance. Further investigation into this issue will be the subject of future work.

TABLE III: Synthesis results for the low-area architecture

Param. Sizes	Freq. (MHz)	DSP48E1s	Slice Reg	Slice LUTs
Toy	307.754	64	19167	11976
Small	293.251	128	37774	23694
Medium	293.822	128	37797	23794
Large	227.388	256	74997	53713

TABLE IV: Average running time of the existing implementations of the integer-based FHE encryption step

Design	Toy	Small	Medium	Large
Low-area architecture	0.016s	0.313s	5.773s	81.550s
High-speed architecture	0.006s	0.114s	2.018s	32.744s
Original scheme[1] [12]	0.05s	1.0s	21s	7 min 15s
FPGA implementation [20]	0.011s	0.306s	7.586s	159.173s

The timings in Table IV are calculated using the clock cycle count and the synthesized design frequencies are stated in Table II and Table III. The read and write operations to external RAM are included within the clock cycle count; two clock cycles are used to read or write each 64-bit word to and from the memory. For example, the clock cycle latency for the multiplication operation is calculated as follows: the multiplication unit requires $2 \times \lceil \frac{w}{16} \rceil + 2$ clock cycles, where 16-bits is the size of the multiplication used within each DSP block and w is the multiplication block width, defined in Table II. In the first architecture, $\tau \times \lceil |x_i|/w \rceil$ multiplications are required. Thus for solely the multiplication operation required in the low-area design, for example the toy sized design requires $158 \times \lceil \frac{150000}{1024} \rceil \times (2 \times \lceil \frac{1024}{16} \rceil + 2) = 3019380$ clock cycles, which takes around 0.0098s.

As can be seen in Table IV, both proposed architectures perform better than the benchmark software implementation [12]. The first and second designs are approximately 3 times and 8-10 times faster respectively when compared to the original results. This matches the speed up factors achieved by other hardware implementations of alternative FHE schemes.

Cao *et al.* [20] have also implemented optimised architectures for the same encryption step (1) targeting the same Virtex-7 FPGA device and using FFT for the large integer multiplication. For comparison purposes, the timing results of the design which fits on the target Virtex-7 FPGA [20] are given in the last row of Table IV. As can be seen from this table, our proposed high-speed architecture performs consistently better than the previous implementations. For example for the medium parameter size, the high-speed architecture is 10.5 times faster than the original software implementation and 3.8 times faster than the implementation using the FFT multiplier [20]. Moreover, our design reduces the number of write operations required by using the Comba scheduling method instead of the schoolbook multiplication scheduling used in [20], which requires intermediate read and write operations. The Comba scheduling method employed in the proposed designs in this work ensures that the partial products are accumulated within the multiply-accumulate blocks, and the only write operations are for the digits of the final result of the multiply-accumulate operation, stated in Equation (1). To the best of the authors' knowledge, no other comparable hardware architectures exist other than those provided in Table IV.

Our designs also use a reasonably small (64-bit) memory interface. These results from Table IV can also be further improved upon, for example, by using a pseudo-random number generator to generate public key values on the fly as

proposed in [12], rather than reading from off-chip memory. This, and both algorithmic and implementation optimisations, such as batching, would further increase the practicality of these schemes.

VII. Conclusion

In conclusion, novel optimised low-area and parallel architectures of the encryption step of an integer-based FHE scheme are proposed and optimally implemented on a Virtex-7 FPGA, achieving speed up factors of around 3 and 10 when compared to the benchmark software implementation respectively. The results achieved illustrate that the use of an optimised architecture to a specific FPGA device which contains specialised DSP slices improves the practicality of the implementation of these FHE schemes and brings them closer to deployment. However further research into optimisations is still required, both at the algorithmic and the architectural level, before FHE schemes can be used in real-time applications. Future work will investigate the use of the Comba scheduling method with the FFT multiplier in order to provide further performance improvements to FHE.

References

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[2] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 18, p. 111, 2011.

[3] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *STOC*, 2012.

[4] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," *IACR Cryptology ePrint Archive: Report 2011/535*, September 2011.

[5] C. Gentry and S. Halevi, "Implementing Gentry's fully-homomorphic encryption scheme," in *EUROCRYPT*, 2011, pp. 129–148.

[6] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography*, 2010, pp. 420–443.

[7] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *EUROCRYPT*, 2010, pp. 24–43.

[8] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," *IACR Cryptology ePrint Archive*, vol. 2011, p. 680, 2011.

[9] ——, "Fully homomorphic encryption with polylog overhead," *IACR Cryptology ePrint Archive*, vol. 2011, p. 566, 2011.

[10] ——, "Homomorphic evaluation of the AES circuit," *IACR Cryptology ePrint Archive*, vol. 2012, p. 99, 2012.

[11] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *CRYPTO*, 2011, pp. 487–504.

[12] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *EUROCRYPT*, 2012, pp. 446–464.

[13] J.-S. Coron, T. Lepoint, and M. Tibouchi, "Batch fully homomorphic encryption over the integers," *IACR Cryptology ePrint Archive*, vol. 2013, p. 36, 2013.

[14] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in *HPEC*, 2012, pp. 1–5.

[15] W. Wang and X. Huang, "FPGA implementation of a large-number multiplier for fully homomorphic encryption," in *ISCAS*, 2013, pp. 2589–2592.

[1]Note, the integer-based FHE scheme by Coron *et al.* [12] is implemented on a single core of an Intel core2 Duo E8400 platform at 3 GHz.

[16] X. Cao, C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "High speed fully homomorphic encryption over the integers," in *Workshop on Applied Homomorphic Cryptography*, 2014.

[17] Y. Doröz, E. Öztürk, and B. Sunar, "Evaluating the hardware performance of a million-bit multiplier," in *16th Euromicro Conference on Digital System Design (DSD)*, 2013.

[18] ——, "Accelerating fully homomorphic encryption in hardware," 2013, draft, Under Review. [Online]. Available: http://ecewp.ece.wpi.edu/wordpress/vernam/files/2013/09/Accelerating-Fully-Homomorphic-Encryption-in-Hardware.pdf

[19] C. Moore, N. Hanley, J. McAllister, M. O'Neill, E. O'Sullivan, and X. Cao, "Targeting FPGA DSP slices for a large integer multiplier for integer based FHE," *Workshop on Applied Homomorphic Cryptography*, vol. 7862, 2013.

[20] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction," *IACR Cryptology ePrint Archive*, vol. 2013, p. 616, 2013.

[21] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM Systems Journal*, vol. 29, no. 4, pp. 526–538, 1990.

[22] Xilinx. (2014) 7 series FPGAs overview. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[23] A. A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, pp. 595–596, 1963, URL: `http://cr.yp.to/bib/entries.html#1963/karatsuba`.

[24] T. Güneysu, "Utilizing hard cores of modern FPGA devices for high-performance cryptography," *J. Cryptographic Engineering*, vol. 1, no. 1, pp. 37–55, 2011.

[25] J. F. Dhem, "Design of an efficient public-key cryptographic library for RISC-based smart cards," Ph.D. dissertation, Université catholique de Louvain, 1998.

[26] P. Barrett, "Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *CRYPTO*, 1986, pp. 311–323.

[27] Xilinx. (2013) 7 series DSP48E1 Slice. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[28] ——. (2011) 7 series FPGAs Memory Interface Solutions. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v1_2/ds176_7Series_MIS.pdf